NAVAL POSTGRADUATE SCHOOL

Monterey, California

THESIS

A DATA DEFINITION LANGUAGE FOR GLAD

by

Kenneth L. Wartick

June 1986

Thesis Advisor: C. T. Wu

URITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited
DECLASSIFICATION / DOWNGRADING SCHEDULE	
ERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
NAME OF PERFORMING ORGANIZATION aval Postgraduate School — 6b. OFFICE SYMBOL (If applicable) 52	7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School
ADDRESS (City, State, and ZIP Code) onterey, CA 93943-5000	7b. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000
NAME OF FUNDING/SPONSORING ORGANIZATION — 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

TITLE (Include Security Classification) Unclassified
Data Definition Language for GLAD

PERSONAL AUTHOR(S)
enneth L. Wartick

. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
asters Thesis	FROM ________ TO ________	86 June 20	92

SUPPLEMENTARY NOTATION

COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	GLAD, Data Definition Language, Graphical Interface

ABSTRACT (Continue on reverse if necessary and identify by block number)

GLAD is an innovative approach to bringing the power of database processing to a larger audience. How this graphical interface approaches this goal is through pictorial representations that convey easily understood concepts to the unsophisticated user. GLAD incorporates many techniques to ease the effort required to effectively use a DBMS but is founded in four basic premises. These principles state that a DBMS interface must be descriptive, powerful, easy to use and easy to learn. This thesis proposes a data definition language (DDL) that both effectively describes the GLAD data model and, at the same time, is in accordance with GLAD's cardinal premises. The database creation process is examined to include incorporating integrity constraints, object relations and object properties. Next, the DDL to (Continued)

DISTRIBUTION/AVAILABILITY OF ABSTRACT [✓ UNCLASSIFIED/UNLIMITED] ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
a. NAME OF RESPONSIBLE INDIVIDUAL Prof. C. T. Wu	22b. TELEPHONE (Include Area Code) 408-646-3391 — 22c. OFFICE SYMBOL 52Wq

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

Block 20 (Continued)

relational database, specifically INGRES, is examined. The thesis concludes with a series of appendices that demonstrate the proposed DDL, a formal specification and proposed guidelines.

A Data Definition Language for GLAD

by

Kenneth L. Wartick
Captain, United States Marine Corps
B.S., United States Naval Academy, 1979

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL

June 1986

ABSTRACT

GLAD is an innovative approach to bringing the power of database processing to a larger audience. How this graphical interface approaches this goal is through pictorial representations that convey easily understood concepts to the unsophisticated user. GLAD incorporates many techniques to ease the effort required to effectively use a DBMS but is founded in four basic premises. These principles state that a DBMS interface must be descriptive, powerful, easy to use and easy to learn.

This thesis proposes a data definition language (DDL) that both effectively describes the GLAD data model and, at the same time, is in accordance with GLAD's cardinal premises. The database creation process is examined to include incorporating integrity constraints, object relations and object properties. Next, the DDL to relational database, specifically INGRES, is examined. The thesis concludes with a series of appendices that demonstrate the proposed DDL, a formal specification and proposed guidelines.

TABLE OF CONTENTS

I. INTRODUCTION 9

A. BACKGROUND 9

B. MOTIVATION 10

C. ALTERNATIVES 13

D. CONTRIBUTIONS AND OUTLINE OF THESIS 15

II. DATA MODELS 17

A. INTRODUCTION 17

B. SURVEY OF SOME DATA MODELS 18

1. The Entity-Relationship Model 18

2. The Relational Model 19

3. The CODASYL DBTG Model 20

C. THE GLAD DATA MODEL 21

III. GLAD's DATABASE DEFINITION 26

A. OVERVIEW 26

B. USER INTERACTION 27

C. GLAD's DATABASE DEFINITION SUBSYSTEM 29

1. Entering the Database Definition Subsystem 30

2. Establishing the Database Shell 31

a. Database Name 31

b. Owner Name 31

c. Permissions 31

d. Password (optional) 32

3. Create Objects 33

a. Object Name 33

b. Object Description 33

c. Object Class 33

4. Atomic Subobjects 34

a. Property 34

b. Integrity Constraints 37

5. Relational Subobjects 38

6. Overview Objects 39

D. OBJECT TO OBJECT RELATIONSHIPS 41

IV. INTERFACE MAPPINGS 43

A. DDL TO THE GLAD DATA MODEL 44

1. The Solid Line 44

2. The Rectangular Box 45

3. The Dotted Line and Double Rectangle Box 47

4. The Small Circle 48

5. The Large Circle 51

B. DDL TO INGRES 52

V. CONCLUSIONS AND RECOMMENDATIONS 58

APPENDIX A: FORMAL SPECIFICATION OF A DDL FOR GLAD 60

APPENDIX B: GLAD DIAGRAM INTERPRETATION 65

APPENDIX C : EXAMPLE CONCEPTUAL SCHEMA 68

APPENDIX D : SAMPLE SESSION 69

PART I: Using the DDL to Create Schema 69

PART II: INGRES Statements to Store Data 85

APPENDIX E: PROMPT FLOW AND SUGGESTED HELP DATA WITH PROMPTS 88

LIST OF REFERENCES 90

INITIAL DISTRIBUTION LIST 91

ACKNOWLEDGEMENTS

I would like to thank the following people.

Dr. C. Thomas Wu for giving me the latitude and confidence I needed while writing this thesis.

My beautiful wife, Lisa, for her encouragement, devotion, love and unusual patience.

I. INTRODUCTION

A. BACKGROUND

Much has been written about the impact of the computer on society. One of the most important contributions the computer has made is to increase productivity. Modern word processing, electronic mail, spread-sheets, communications networks, and database management systems (DBMS) all owe their existence to the computer revolution. It was not so long ago that those who used computer equipment were limited to the data processing professionals. Today, elementary school students to secretaries to business executives daily use computers in varying degrees of sophistication.

The user population base has grown rapidly to encompass almost anyone who uses information in their work or leisure. This new generation of computer users have challenged the notion that computers have to be difficult to use. People want computers to solve problems, not to unduly complicate the solution process. In order to appeal to this diverse range of users, computers must be easy to use and be useful. These two attributes; usefulness and ease of use, are the driving premises to this thesis and the creation of a new database interface package called ***G***raphic ***LA***nguage for ***D***atabases (GLAD). Computer systems that master these two criteria will be the most successful.

If a system is hard to learn. of those capable of mastering the system few may be willing to expend the time and energy at the expense of other activities. And if a system is not useful then it simply will not be used.

One of the fastest growing application areas using computers is the manipulation of databases. The system used to create, modify, and manipulate these large tracts of data is called a DBMS. Many corporations are now viewing their databases as corporate assests and as a tool to gain the competitive advantage over their rivals in the marketplace. The use of this information can range from trimming internal costs and providing the tools managers need to make better decisions to the difference between profit and failure. The DBMS is not a passing fancy but a crucial instrument to the effective administration of many corporations. The proliferation of personal computers and DBMS's shows the need for good man machine interfaces but raises the question of whether or not whats available is what is wanted.

B. MOTIVATION

The evolution of the DBMS has been slow. For much of the computer's history, file processing has dominated the management and manipulation of data. File processing is still valid for many applications but the DBMS is growing in popularity. While the DBMS will never fully replace file processing the trend towards the DBMS is motivated by [KROE83] :

* More Information from the Same Amount of Data

* New Requests and One-of-a-Kind Requests More Easily Implemented

* Elimination of Data Duplication

* Program/Data Independence

* Better Data Management

* Affordable Sophisticated Programming

* Representation of Record Relationships

While these are good reasons to use a DBMS, it is equally important to give the unsophisticated user access to the power of a DBMS. The unsophisticated, or naive, user can be anyone whose expertise lies elsewhere than with computers.

As the value of data grows in importance so does the need to effectively use this data. Any DBMS must perform a wide variety of functions ranging from security and integrity of data to querying the database and recovery procedures. Many of these functions are transparent to the user and do little to influence the user's ability or desire to utilize the DBMS. The two factors that matter most to any user are, again, usefulness and ease of use. These two factors are directly addressed by the proposed database interface package: GLAD [WU85]. This thesis is directly concerned with that portion of the database package that defines the database characteristics, specifically the data definition language (DDL), while still keeping the basic premises of usefulness and ease of use.

Some of the disadvantages of the DBMS have been its cost. the drain of the CPU resources demanded by the DBMS. and recovery and concurrency problems. There exists ongoing research to alleviate these drawbacks but these specific problems have little to do with user acceptance. This is not to say that these are not important issues since they obviously are, but, the technical aspects of concurrency control are transparent and irrelevant to the casual user. Indeed, this class of user should not worry about these considerations. What the user wants is ease of use and usefulness.

Ease of use encompasses a wide variety of areas. Performance, help facilities, clarity and other factors affects it. Ease of use demands that the user not memorize a lot of commands nor should they have to type redundant data. Whenever possible, the database interface should display applicable choices so selection can be made by using an electronic mouse or selection from an ordered list. This would reduce training time and generate greater acceptance by a larger population base.

In order to be useful, any DBMS must be capable of precisely capturing real world semantics. The chosen data model must represent real world data and relationships. Furthermore, this data must be valid, secure and retrievable in different manners. The retrieval process should allow users a variety of ways to achieve the same results.

The data model must allow for change if it is ever to be useful. Multiple views of the database schema are mandated. Many other factors apply but the premise is clear that usefulness means the system must model the real world. be powerful yet allow maximum user tolerance. These are important goals that have been typically compromised for one reason or another.

The best motivation for any database system is to achieve popularity on its merits rather than on advertising. The best database interface system must strive to address the need to make database access user friendly to even the most computer illiterate individual.

C. ALTERNATIVES

A great deal of papers have been written in the database area. Of these papers. perhaps the most original and profound research has been by M. M. Zloof in his Query-By-Example (QBE) [ZLOO77]. His ideas relate directly to posing queries to a database without the user needing much knowledge of the database naming conventions. By using a two dimensional skeleton table. the user specifies queries by designating the criteria for database searches in the computer displayed tables containing the names of the data items as headers. The most important aspect of QBE is not the complexity of the queries that can be accomplished but rather the concept of presenting information about the database in a clear and understandable way. Rather than forcing the user to memorize data names or enter queries in a rigid format. QBE has gone a long way towards true *user friendliness.* Unfortunately. QBE is limited in its ability to describe database schemas and relationships. In this sense. QBE lacks *descriptiveness.*

Other suggestions and techniques have been proposed to decrease the complexity of the user-database interface. Natural language interfaces have been discounted for their lack of power [PETR76]. However, true usefulness implies power and descriptiveness. Ease of use implies not only easy manipulation of data and adequate help facilities but low training time as well. It seems that those systems that are powerful offset this advantage by being unduly complex. Fourth Generation Languages (4GL) claim progress in this area but tend to lack the graphics capability to permit database schema information to be represented in an easily understood form.

This thesis is primarily concerned with the user's interaction with data definition while creating the database schema. GAMBIT [BRAG84] is an excellent tool in database definition that incorporates semantic integrity constraints as well as interactive modeling. GAMBIT is intended to be used by the database administrator. GAMBIT requires intimate knowledge of database techniques and is not useful for those needing only to create simple databases. Nor does GAMBIT fulfill all functions of the database administrator as opposed to those shared with naive users.

GLAD's data definition subsystem and DDL should keep the characteristics of descriptiveness, power, ease of use and ease of learning even though the primary users are those with database administrator status. A single coherent database interaction system is needed that concerns itself with not only data definition and queries but the four characteristics just mentioned as well. This thesis' proposal for GLAD's data definition subsystem and DDL attempts to satisfy these requirements.

D. CONTRIBUTIONS AND OUTLINE OF THESIS

Chapters 1 and 2 cover introductory material. This chapter has presented the need for a DBMS interface that is powerful, easy to use, easy to learn and describes a complex world in a complete manner. The proliferation of computers and the growing importance of databases to an increasing and diverse population was discussed. The overall goal of this thesis is to propose the needed functions and techniques of a database data definition interface.

Chapter 2 explores the concept of the data model. Several prevalent data models are cursorily examined to include the Entity-relationship, relational and CODASYL models. The data model for GLAD is explained to include supported concepts, basic terminology and the model's graphical representations.

Chapter 3 covers the functions needed to create a suitable environment for creation and maintenance of databases. Issues of data access, security, integrity and constraints are introduced. The framework of the proposed data definition language is introduced. GLAD's database schema concepts are shown to be supported by the proposed DDL.

Chapter 4 carefully details the proposed DDL's constructs to include not only the syntax but semantics as they are related to GLAD's data model. The chapter also explores some of the relationships shared by the GLAD interface and the underlying database management system. INGRES in this case.

The last chapter summarizes the contributions of the thesis and establishes possible future work in this and related areas.

The appendicies provide a formal specification of the proposed data definition language as well as a detailed sample session, suggested guidelines and other explanatory material.

II. DATA MODELS

A. INTRODUCTION

The purpose of the data definition language is to translate real world objects and relationships into a form understood by the computer that still preserves the original data characteristics. The representation of this information to the user is done in the form of a data model. The view of the database as seen by the database administrator is called the *conceptual* schema. The *subschema* is simply a subset of the conceptual schema. The subschema is a partial view of the database as designated by the database administrator. Finally, the view of the database as seen by the computer is called the *internal*, or physical view. These three views have been proposed and endorsed by the ANSI/X3/SPARC Committee [TSIC78].

The design of a database is an intuitive and sometimes artistic process. The database administrator can use many analysis tools ranging from entity-relationship and data flow diagrams to hierarchical input processing and output (HIPO) charts. However, when the database administrator feels that the real world semantics have been captured either mentally or on paper, the final translation into the computer must take place. It is important that all the relationships and data elements be preserved and, more importantly, that the computers representation of this data be easily understood by the user. The data model. then. is a pictorial representation of the database that can be understood by both computer and man with some translation procedures. The purpose of the data model, however is ***not*** to aid in the interactive design of the database schema. While this is possible for the experienced database administrator, it is not considered an advertised feature of GLAD.

Choosing a data model is an acceptance of compromises from one data model over the next. Depending on the application at hand, one particular model may be better suited than another from the users point of view but may change with the next application. Also of importance is the difference of desirability of a data model from the perspective of what is considered important. If performance is the dominant criteria then one model may be chosen, but if descriptiveness or power is foremost then other models might be judged superior. Because of these trade-offs. it is beneficial to examine some of the prevalent data model's characteristics.

B. SURVEY OF SOME DATA MODELS

A considerable number of data models have been proposed and much fewer implemented. The following data models are briefly examined since they exist in actual implementations or have been subject to considerable study.

1. The Entity-Relationship Model

In the Entity-relationship (ER) model [CHEN76], a database schema is represented by a graph called an entity-relationship diagram. This model's major characteristic is the existence of attributes, each mapped from an entity set or a relationship set into a value set or a Cartesian product of value sets. This model's value lies in its ease of understanding and also as it is the basic foundation to a great many models.

Basic to the ER model are *entities* that represent things or concepts. Thus "chair". "emotions" and "receipts" can be entities. Entities are described by its properties called *attributes.* Attributes associate a value from a domain of values for that entity. An attribute. or set of attributes. that uniquely identify or determine an entity among a set of entities is called a *key.* A relationship among entity sets is simply an ordered list of entity sets. The *functionality* of a relationship numerically signifies how entities are viewed through a relationship and can be either one to one. one to many. or many to many.

Typical ER notation puts entities into rectangles, named relationships as linked diamonds between the two entities and functionality as labels on the lines between the entities. Attributes are usually not listed since they tend to clutter the database view. The ER model is excellent as a logical database model since it preserves data independence but it fails to capture the semantics of complex interrelationships between entities.

2. The Relational Model

One of the most important data models is E. F. Codd's relational model [CODD70]. The center of much academic interest and commercially available, the relational model has been lauded as a good interface between man and machine.

The beauty of the relational model is its simplicity without sacrificing power. Data is presented as a table. The rows in the table are called *tuples* of the relation while the columns are the relation's *attributes*. The significance of the relational model is not that the data is arranged in relations but that the relationships are considered to be implied by the data values. This flexibility of carrying relationships in data means that relationships need not be predefined.

The relational model is well understood and has its very roots well founded in mathematics. The model's characteristics can be described through relational algebra and relational calculus [ULLM82]. The relational model is powerful. To link two data elements requires only that they share a common attribute. This linking is called a *join*. The relational model effectively supports many to many relations and enjoys a growing popularity.

3. The CODASYL DBTG Model

The CODASYL DBTG (Conference on Data Systems Languages, Database Task Group) data model is the oldest of the data models. This model is closely tied to the physical characteristics of the data and consists of *data items*, *records* and *sets* to describe its data. A set is a named collection of one or more record types where one record is the owner and the others are members. Relationships are built into the schema and are not easily changed. The CODASYL model can be very efficient in certain applications that deal with structured relationships but have difficulty in implementing with any degree of efficiency the many to many functionality.

Since the CODASYL model is not as logically oriented as the ER model or the relational model, it requires a more complex translation from real world to conceptual schema but less conversion to the internal schema.

C. THE GLAD DATA MODEL

The model chosen for use with GLAD is in accordance with GLAD's cardinal premises as set forth by [WU85]:

1. It must be descriptive. The schema representation conveys real world semantics unambiguously and clearly. The schema preserves data properties and permits the user to ask for help on any element or representation.

2. It must be powerful. The data definition language must be capable of capturing real world semantics precisely. The DDL must completely support GLAD's data model allowing the model to be constructed in any arbitrary yet valid form. The DDL must incorporate an extensive error checking mechanism that detects both logical and syntactical errors.

3. It must be easy to learn. No, or very few, commands should have to be memorized and syntax should be kept simple. The flow of user operations should proceed in a logical manner that causes an effect when any action is taken. The schema must mimic currently accepted representations of objects or entities.

4. It must be easy to use. The data definition language must not be cumbersome but allow the user to create the schema in an unordered fashion. Use of an electronic mouse would reduce the necessary keyboard typing to a minimum.

In addition to these basic requirements, the DDL must support generalization, aggregation and classification as outlined in [WU85]. It is clear that the GLAD date model must resemble a model closer to user friendliness such as the ER or relational model as opposed to machine suitable as with the CODASYL model.

In fact. the GLAD data model closely resembles the ER model with some extended enhancements. The basic element is the *object*. In this sense the object corresponds with the entity in the ER model in that the object can be both intangible and tangible but the GLAD object has its own unique characteristics as we will see. An object can be composed of other subobjects. Because of this composition, we say an object is an *aggregation* of subobjects. This aggregate object is represented as a named rectangle in the GLAD schema. The aggregate object can be expanded to reveal its components. The components can be *atomic* in that an atomic object is composed of only one system defined or user defined *base* object. A non-atomic object is either a *relation* or another generalized object.

The normal relation between two aggregate objects is specified by the "LINE_TO" operator. To signify a one way relation. a subobject of one aggregate object will be the name of the relation and the name of the aggregate object sharing the relation. Since each relation is always two way. it is not necessary to specifically designate the relation in each object although it would not hurt. Another relation supported under GLAD is called the *disjunctive* association. The disjunctive association means that an *instance* of an aggregate object at any time *t* can be associated with object1 or object2 or object*n* but not more than one within the group. The disjunctive association is characterized by listing two or more mutually exclusive objects using the "EITHER" operator instead of just one object as was the case when specifying a normal relation between two aggregate objects. If an object can have two or more normal relations between itself and others then each relation must be specified with its own "LINE_TO" operator. The disjunctive association is graphically represented by a small circle between the aggregate object and the relational lines to the disjunctive objects.

GLAD supports *generalization.*. When individual objects such as *faculty, secretary* and *technician* can be grouped together to form *employee.* We call *employee* a generalized object. Conversely. *faculty, secretary* and *technician* are *specialized* objects. Generalization abstraction is characterized as an IS_A relationship (*faculty* IS_A *employee).* In this case *employee* would be an aggregation of objects shared by the other three and would specify this unique relationship by the "IS_A" operator. An example is shown in figure 2.1.

Since *faculty* IS_A *employee,* the subobjects and relations of employee are equally shared by *faculty* as if they were listed within *faculty's* definition. *Employee,* however, ***IS_not_A*** *faculty.* so that *employee* does not acquire the subobjects of *faculty.* In other words. the inheritance of relations and properties only extends from specialized to generalized object and not vice versa. There does not have to be anything in *employee* that designates it as a generalized object, but when *faculty* is defined by virtue of its IS_A operator. it designates itself as a specialized object and *employee* as a generalized object. Although it is possible for a generalized object to have only one specialized object relation it would make

```
object name: EMPLOYEE.
property: EMP.TYPE EITHER (FACULTY,SECRETARY,TECHNICIAN).
          property: BELONGS_TO LINE_TO (CAMPUS).
          property: SSN.
          key?: Y.
                    property type: NUMERIC.
                    numeric length: 9.
                    numeric constraints: MUST FILL.
          property: EMP.NAME.
          key?: N.
                    property type: TEXT.
                    text length: 15.
                    text constraints: MUST EXIST, JUSTIFY(R).
          property: END.

object name: FACULTY.
          property: SSN.
          same as in object "employee"?: Y.
          key?: Y.
          property: IS_A (EMPLOYEE).
          property: TENURE.
          key?: N.
                    property type: ENUMERATION.
                    enumeration list: NONE, SOME, FULL.
                    enumeration constraints: .
          property: DEPT_BELONGS_TO LINE_TO (DEPT).
          property: END.

object name: END.
```

Figure 2.1 - Generalization Example

more sense to group both sets of properties under one object in this case. Once an aggregate object becomes a generalized object it is then graphically represented as a named *double* rectangle in the schema. The line from a generalized to specialized object is drawn as a *dotted* line to signify the special relationship that is shared between them.

Classification is a term that simply relates databases to the object that it belongs with. Throughout this thesis. we call each data item of an object a *member* of that object. In the relational model. a data item is merely an attribute value.

The GLAD data model is unremarkable in that its schema makes use of only three symbols (rectangle. nested rectangle and small circle) and two lines (dotted and solid). What is remarkable is that the schema supports generalization. aggregation and all ER model attributes. A further enhancement is GLAD's ability to organize the database schema using overview objects as discussed in the next chapter.

III. GLAD's DATABASE DEFINITION

A. OVERVIEW

GLAD can be viewed as two subsystems. The view seen by the common user is limited to the manipulation of physical data. This class of user, however, cannot alter the underlying physical structure of the database itself, such as relationships between objects, only the data values. The structure of the database, or metadata, is manipulated only under the second view of the database. This second view, under the purview of the database administrator, is concerned with the creation and updating of database characteristics as well as other administrative matters such as database access authority.

Keeping two separate views is considered essential to maintaining the integrity of the database. To combine the two views would burden the the common user with additional needed knowledge to understand the system as well as jeopardize the database itself. The user of the GLAD database definition subsystem must be the database administrator only. Although it is not sound practice to interactively create the database as it is with the formulation of queries, GLAD does allow creative formulation of schemas if so desired. It is important. as mentioned before. GLAD is not intended as an interactive modeling system. The creation of a database requires thoughtful planning and analysis of users needs well before attempting to implement the DBMS.

GLAD provides the database administrator with the unique advantage of viewing the database structure diagram as it is being created. In this way, the intended final view of the database can be compared against what is actually entered as objects and relations. These partial views can confirm or give question to the database administrator's view of the data and that actually presented by the GLAD system. The errors generated by GLAD can alert the database administrator to oversights in the database creation process.

B. USER INTERACTION

From this point on, the user is considered to be the database administrator unless otherwise specified.

All of GLAD's inherent advantages mentioned in the preceding chapters must be applied to the data definition subsystem as well. The DDL must support GLAD's data model completely and still keep with the spirit of user friendliness even though the user is expected to be the database administrator. In this sense, the experienced database administrator should be able to quickly create complex database schemas and, at the same time, the novice database administrator can implement rudimentary systems as well. This flexibility allows the user to implement personal databases with little effort although GLAD is intended to serve the multi-user community. When the database administrator understands what the users want and can "visualize" the database, then GLAD can expedite the creation process.

Once the database's characteristics have been developed, the process of creating the physical database itself should be in accordance with GLAD's four intrinsic characteristics [WU85]:

1. It must be descriptive. The database creation process should allow for a graphical display of current database elements and relations. While this can be performed in a variety of ways, the use of windows, as stipulated in GLAD's query processes, is preferred. As each object is created, a provision for display of the object in a graphical representation should be provided. To carry this one step further, the database schema should be allowed to be displayed at any time. If the database is only partially defined then the progress should be evidenced by the schema's representation.

2. It must be powerful. GLAD must allow any database schema that can be represented by entity-relationship diagrams. In addition, the concepts of aggregation, generalization and specialization are to be supported.

3. It must be easy to learn. The creation process should be near self explanatory to the database administrator. This database administrator is not considered a naive user. If an individual knows enough to develop a database schema then the process of entering this information, via GLAD, should require minimum training.

4. It must be easy to use. Keyed entry or use of a mouse is provided. A relation between two objects can be declared by movement of the mouse over the object's graphical representation. Once the objects are selected, the GLAD program should automatically list the attributes with identical field types that are common to both objects for selection as the connecting attribute(s). If a mouse is unavailable, the entries can be keyed with the same results.

By incorporating the above characteristics, GLAD will allow the database administrator the ability to quickly create and verify a database. The verification is an important quality. The database administrator can cross check the final GLAD database schema diagram with the diagram drawn in the database's planning stages. If they match, then a high probability exists that what was created is what the database administrator had originally envisioned. If a discrepancy exists, then it becomes readily apparent and the error quickly isolated.

C. GLAD's DATABASE DEFINITION SUBSYSTEM

Creating the database attributes is perhaps the most labor intensive operation within GLAD. This is expected and considered normal in any database system. The first step in entering the database definition subsystem is a routine clearance check through a password system. Once the user is cleared the database creation process can begin.

The GLAD system is centered about objects and the interrelationships among the different objects. Each object is named and its properties defined over the available type domains. Each property of an object, in turn, may itself be a distinct object. The relationships between objects as attributes can get quite complex.

Only a small number of objects can be displayed on the screen as graphical objects at one time. The objects that are displayed when first viewing the database are those that belong to the database schema overview diagram. To view all other objects will then require a "superior" object to be expanded. This expansion is a GLAD command. The expansion process can proceed until an atomic, or "primitive", object is encountered. As a direct result of this process, all of the top-level objects need to be identified as such at the time of their creation.

Choosing these overview objects requires much thought and consideration. If too many objects are displayed, the schema might become confusing and cluttered. Too few objects might remove any perspective of the world that the database is attempting to model. Another drawback of too few objects is that the user may have to expand every object on a constant basis to reveal those objects and attributes most heavily used.

One of the advantages of GLAD will be its ability to restructure the top-level view of the database without necessitating the restructuring of any physical data. This is a dynamic process that provides great flexibility in meeting individual user's needs. The following discussion takes the reader through the steps needed to create a working database.

1. Entering the Database Definition Subsystem

The database definition subsystem can be entered in one of two ways. As in many menu driven systems, each subsystem is listed as a numbered option. This is not the desired approach since few users would require use of the database definition subsystem. Listing the subsystem as an option would violate the cardinal premises on which GLAD was founded by presenting unneeded information.

The second. and preferred. method is to enter the subsystem by issuing a command to the resident operating system. This may be somewhat less convenient for the database administrator but the thrust of GLAD is to make things easier on the common user. Therefore, every effort, no matter how small. has been made in this direction.

2. Establishing the Database Shell

Prior to defining the database's schema. the database's passwords, availability and owner must be specified. This creates the necessary environment for subsequent updates to the particular database. The information on a database must be kept in the GLAD system's files. The information held in this file must. at a minimum. be as follows:

a. Database Name

This must be unique to the user. GLAD must use a system of user login name and database name. This is important so that meaningful names can be assigned without regard to other user's choices of names.

b. Owner Name

The owners name is automatically entered by GLAD. This information is known by the operating system and by GLAD through the login process.

c. Permissions

Other users are granted an overall database permission. This permission applies to the physical data itself and not to the metadata. Only the owner of the database. the database administrator. can alter the physical database schema itself. The basic data permissions include:

1. Retrieve. Allows users to make queries against the database.

2. Add. Allows new data to be entered. In the relational model this would mean that the user can add new rows to existing tables while in the CODASYL model it would allow adding records.

3. Delete. Allows existing data to be deleted in the same manner as the "Add" permission above.

4. Change. Allows changing data values only.

5. All. Includes all permissions.

These permissions are the default values for all data objects. These values can be changed for each object on a case by case basis to allow greater flexibility. The individual permission differences can be stipulated at the time that the different subschema are defined.

d. Password (optional)

Rather than specifying the users who are granted access to the database it is more manageable to incorporate a password system. In future implementations a reiterative process of permissions and passwords could allow a diversity of access constraints to the database using the basic modules already presented.

This phase of the database development can easily be implemented through a series of prompts and graphic aids. For example, permissions can be displayed and selected by a mouse.

3. Create Objects

This phase of database development is the most difficult unless each object is well planned. An object is a named entity composed of an aggregation of (sub) objects. Objects are either *atomic* or aggregate objects. Atomic objects consist of only one system defined or user defined *base* object. Furthermore, an atomic object can have integrity constraints imposed such as allowed subranges. The following operations are available when creating *any* object.

a. Object Name

Some system limitation must be imposed so that the name will fit into the object's displayed box (e.g.: it graphical representation). The object name must be unique among all other objects within the database. This step must be performed and, indeed, should be prompted as the first step in the object creation process rather than presented as an option. The result of successfully completing this step will be the addition of an object box in a window so that it can be referred to latter if needed using the mouse.

b. Object Description

This is an option and can be skipped. Allowing an object to be described can be useful when a user selects the *HELP* command while formulating queries. With this in mind, the database administrator would describe the object in terms most useful to the common user.

c. Object Class

There are three cases that need to be specified. If the object is *atomic*, then constraint and type information about the object is needed. If an object is *non-atomic* then a relationship type must be established between other objects. The relationship, as we will learn, can be a disjunctive association, a normal relation or the designation of a specialized to generalized object relation. A primary difference between the two is the amount of information needed to specify the property. The relationship properties typically need only one line to full characterize itself while an atomic object may require four or more separate specifications to characterize itself. The third object class is the overview type. This classification is simply a grouping of objects under one heading for database organizational purposes. The object's classification is determined by its syntax alone.

4. Atomic Subobjects

If an object is atomic then it possesses no relationships and has a specific delineated property. Atomic objects can be thought of as an attribute while the characteristics of this attribute are described by its properties. Except for the overview object, aggregate objects must specify key values and these key values must be atomic objects. These keys are essential to the effective internal storage of the object's data members.

In addition to its name, description and class, the atomic object must be defined by its properties and integrity constraints.

a. Property

The type of an object is not necessarily system dependent. The user can specify the *logical* type, such as "integer" independent of its actual internal representation. The following data types can be displayed once an object is determined to be atomic for selection.

(1) Text. If text is specified, then a maximum length must also be specified. Integrity constraints, such as subranges, are not applicable to text, however, constraints such as left justify can.

(2) Integer. Integers are whole numbers whose range is wholly system dependent. Integers can be constrained by subranges and conditionals (< 100). Additionally, integers can be coerced into the floating point type in calculations.

(3) Float. We adopt te traditional definition of floating point numbers as defined by the following Backus Naur Form (BNF) listing:

[+|-] {<digit>} [.<digit>{<digit>}] [e|E [+|-] {<digit>}]

As with the integer type, float can be constrained by subranges limited to the precision of the underlying machine. Float cannot be coerced into integer.

(4) Date. Since dates are so common. a built in feature might include a Date type as used in INGRES. GLAD could display several formats and allow the user to select the appropriate one. For example:

dd-mmm-yyyy
-or-
mm/dd/yy
-or-
16 January 1957

Date can be constrained by subranges or enumeration of any individual component.

(5) Money. As with date. money is a common enough database item that. for sake of convenience, a built in type is included. Money is a subrange of float with a precision of two decimal points. Allowable constraints for float apply.

(6) Enumeration. This is user defined and may be subject to implementation limitations. An example of enumeration would be to list the allowed values for a "color" object as ('RED','BLUE','GREEN'). This is much the same way as Pascal's enumeration method. Ways to enumerate objects range from typing each one in to specifying a file where the different values are contained. For the present, the values are to be listed.

(7) Boolean. Boolean is either true or false. It has its own type since it is commonplace and lends itself to efficient internal storage.

(8) Numeric. The numeric property is a digit subset of the text type. This is useful for such objects as social security number that consist of numbers but are not subject to arithmetic operations. Numeric objects can hold any number of digits unlike the integer type that has an absolute range imposed on it by the underlying hardware. Numeric has constraints shared by both text and integer.

b. Integrity Constraints

Integrity constraints apply to the adding or updating of the physical data. The type of constraint imposed on a data element can be of two forms. The first is similar to the INGRES "Define" QUEL command. In this environment, a data element can be assigned a different permission than assigned to the database and subranges can be stipulated. For example, if the object is "IQ" and is type integer, the following could be designated as constraints:

permission = all, 200<= range <= 50

In the graphics interface, a listing of all permissions (the default might be shaded) and allowed operations might be shown for the object's type. This, however, is not a immediate function of the DDL since permissions are decided when creating subschema views of the conceptual schema. The proposed DDL is for creating the conceptual schema as seen by the database administrator. Subschema definition is an aspect that will encompass a separate aspect of GLAD that is not defined in this thesis.

The second constraint mechanism. which is incorporated within the proposed DDL, provides rudimentary data entry checking. The list of allowable operations would be a function of the object's type and might include:

* must exist
* justify (R/L)
* pad (blanks/zeroes)
* must fill (no blanks)
* must have sign (+/-)
* must begin with (character/number)
* no special character allowed

5. Relational Subobjects

The relational object defines a relation from an aggregate object. The aggregate object is necessarily a composition of atomic subobjects (some or all of which form a key) and has one or more relational properties that show the relationship ‹ the aggregate object with another aggregate object. When two aggregate objects are joined by specifying a relationship, it is imperative that they have a common property definition. Since there is no requirement that this shared property have the same name the only error checking that can be performed is a check that the two aggregate objects share a common atomic subobject definition. This is necessary for two reasons. First, there is no assurance that the two aggregate objects cannot share more than one common attribute that could possibly link the two and. secondly, no checks for dependencies or normal form are incorporated into the design of the DDL at this time.

As mentioned before, relations can be of three varieties. The disjunctive association, the normal relation and the specialized to generalized relation can be achieved. If an object contains no relationships, only atomic attributes. then another generalized object must designate the first object with a relationship or else the schema would contain disjoint objects. If an object specifies a relationship with another then there must exist a common element between the two. Generally. this common element would be the object's key or a subset of the key but not necessarily. The link between the two need not have the same name but must be the same type(s) and lengths.

It is possible to create an object that is neither atomic or relational. This object is special to the database schema and is used to logically group database objects for display on the user's terminal. This object is called an overview object.

6. Overview Objects

The overview object has no atomic attributes and has two or more aggregate objects contained in its definition. This object's sole purpose is to better organize the presentation of the database schema. The database administrator may decide that the database is better presented if portions of the database are grouped under an artificial name to streamline the schema.

For example, if a government database is concerned with assets possessed, some of the top-level objects might be "MILITARY ASSETS", "NATURAL RESOURCE ASSETS", "FINANCIAL ASSETS", etc, refer to figure 3-1. Under "MILITARY ASSETS" might be "AIRCRAFT", "SHIPS", "TANKS", "MARINES", "SOLDIERS", "SAILORS" and "AIRMEN". Like sub-objects are contained in the other assets objects. Already, the schema is cluttered and it becomes unclear how to best present the database schema at the top-level. One solution would be to artificially group "AIRCRAFT", "SHIPS", and "TANKS" under "OTHER" and "MARINES" etc., under "PEOPLE", see figure 3-2.

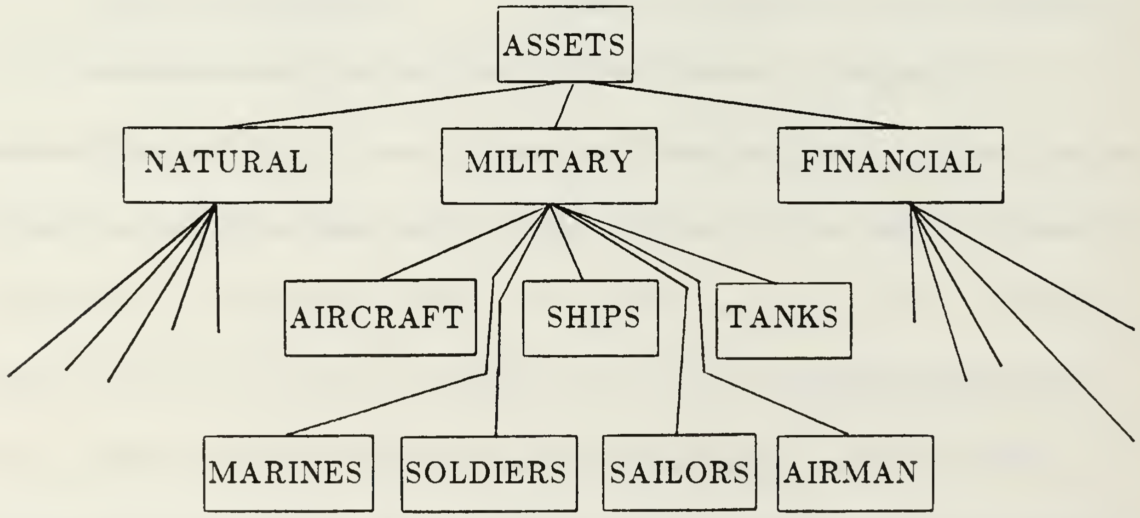


Figure 3.1 - Asset Database

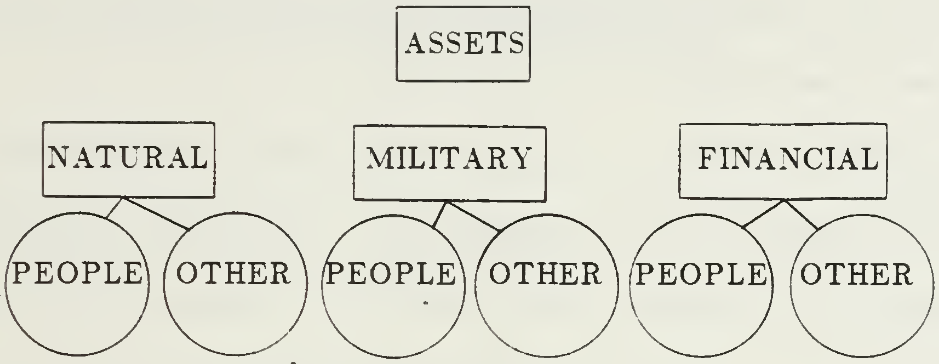


Figure 3.2 - Revised Asset Database

Neither "PEOPLE" or "OTHER" have attributes but the overall database schema can now be presented as each major asset category with two sub-objects (people and other), even though there is no data associated with either object.

D. OBJECT TO OBJECT RELATIONSHIPS

Up to this point a lot of detailed information has been presented describing the relationships of the schema elements and the DDL semantics necessary to implement these concepts. To summarize this chapter it is appropriate to recapitulate the relationships shared among GLAD's objects.

The overview object is composed of any number of aggregate objects. The aggregate object is an entity with certain properties. The aggregate object, which may or may not be either specialized or generalized, is composed of both atomic and relational subobjects. Atomic subobjects contain property specifications, subobjects relate the aggregate object to another aggregate object with either a regular relation. a disjunctive association or as a specialized aggregate object to a generalized aggregate object.

IV. INTERFACE MAPPINGS

In this chapter we show, informally, that the proposed DDL bijectively supports GLAD's data model. Also informally, a partial mapping of the DDL constructs will be mapped to INGRES' QUEL commands. It is beneficial to describe the functions of the GLAD interface first.

What the interface is to accomplish is evident. The interface receives input from the user, *processes* this information in some manner and depending on the work to be performed, the interface either passes the job to the underlying DBMS via the operating system or performs the job itself. It is important to realize that the interface does not merely rearrange the command structure to perform the same tasks but *adds* functionality, descriptiveness, power or friendliness to the underlying DBMS. Of course a certain amount of modification must be performed to the interface to support a different model's implementation. Others use the term "interface" and "translator" interchangeably. We will not. This is because models are called by one name yet implemented in various manners and because each generic model has shortcomings that need to be covered by the interface in varying degrees.

The easiest mapping to show is from the DDL to the GLAD data model since the language was specifically designed for this purpose.

A. DDL TO THE GLAD DATA MODEL

The GLAD data model consists of a number of pictorial objects representing a specific meaning. It must be shown that every object and GLAD concept is supported by a DDL construct or sequence of DDL constructs. Appendix A contains the complete formal specification for the proposed GLAD DDL. From C.T. Wu's papers on the GLAD system, it is clear that the data model supports aggregation, generalization, specialization, the disjunctive association and classification. In addition to these, this thesis advocates the use of the overview object for schema representation. The explanation of how an object is created is kept to a minimum and illustrated with an example.

1. The Solid Line

The solid line represents a named or unnamed normal relation between two aggregate objects. Nothing is implied by this relation other than the two aggregate objects share a common property definition. The solid line is drawn only once even though both aggregate objects might specify each other within a relation definition. If this were not the case then two lines might be drawn and so a hierarchy of line management must be imposed. Within this line management hierarchy the dotted line has precedence over the solid line if ever they should both be specified between the same two objects.

The minimum syntax to specify a solid relational line is:

```
<object_name>.
         <reg_rel_name> LINE_TO (<object_name>).
```

Remember that the other relation types are not solid lines but a variation on it.

A simple example of the normal relation :

```
object name: CUSTOMER.
description?:.
        property: OWES LINE_TO (ACCOUNTS_RECEIVABLE).
        description?:.

                .
                .
                .
```

Within CUSTOMER's definition, the command operator "LINE_TO" actually creates the relation between CUSTOMERS and ACCOUNTS_RECEIVABLE. This is not the only requirement for the relation to exist since a valid key property and, more importantly, the named aggregate object to which the line is to be drawn must exist. Two types of error checking occur. Syntax error checking is immediate and will detect an error such as a misspelled operator or an invalid name. Syntax checking is fully guided and controlled by Appendix A's specifications. Cohesion, or logic, checking is a validation of all relations such that if a name is used as an object name and no such object has been created an error will result.

2. The Rectangular Box

The rectangular box is the representation of the aggregate object. The aggregate object is a composition of atomic subobjects and zero or more relations. The DDL syntax to create the minimum aggregate object are:

```
<object_name>.
        <reg_rel_name> LINE_TO (<object_name>).
        <atomic_name>
                <property_type>
        END.
```

The relation used is the normal relation but might have been the specialized to generalized relation as well. Also note that we stated before that if an object did not specify relations then another object would have to name this object within a relation within its own definition or else the object without the relation would be disjoint to the schema. When a one way relation is specified, the inverse relation is automatically assumed. The following example illustrates the combination of prompting and supplication of answers to quickly create the aggregate object.

```
object name: EMPLOYEE.
description?:.
        property: BELONGS_TO LINE_TO (DEPT).
        description?:.
        property: SSN.
        description?:.
        key?: YES.
                property type: NUMERIC.
                numeric length: 9.
                numeric constraints: MUST_FILL.
        property: EMP.LAST.NAME.
        description?:.
        key?: NO.
                property type: TEXT.
                text length: 20.
                text constraints:.
        property: END.
```

3. The Dotted Line and Double Rectangle Box

The specialized to generalized relation requires a minimum of three aggregate objects to be created. These objects bear a special relationship to each other in terms of real world semantics but the syntax of the generalized object does not reveal this. The syntax necessary to achieve this relationship is found only in the specialized object as shown:

```
<object_name>.
        IS_A (<object_name>).
        <atomic_name>
                <property_type>
        END.
```

The following example better illustrates the generalization concept:

```
object name: CORVETTE.
description?:.
        property: IS_A (AUTOMOBILE).
        description?:.
        property: VIN.
        description?: VEHICLE ID NUMBER.
        key?: YES.
                property type: NUMERIC.
                numeric length: 14.
                numeric constraints: MUST_FILL.
        property: OPTIONS
        description?:.
        key?: NO.
                property type: ENUMERATION.
                enumeration list:  LEATHER, 4BBL, STEREO, PIN-STRIPING.
                enumeration constraints:.
        property: END.
```

In this example, CORVETTE is a specialized object of AUTOMOBILE. We assume that AUTOMOBILE carries other properties like engine size, color, interior etc.. that are common to all cars. We further assume that other specialized objects such as CHEVETTE and ACCORD exist that are specialized objects of AUTOMOBILE. This example will create a dotted line between CORVETTE and AUTOMOBILE and will further clarify the relationship by designating AUTOMOBILE as a double rectangle. If AUTOMOBILE has already been designated a generalized object by, say CHEVETTE, then no change is made to AUTOMOBILE other than it is joined to CHEVETTE by a dotted line.

4. The Small Circle

If an aggregate object contains a disjunctive relation with two or more other aggregate objects then the relation is designated by a small circle attached to the outer box of the aggregate object with lines to the other objects. This disjunctive relation is shown by:

```
<object_name>.
        <reg_rel_name> EITHER (<object_name>,<object_name>[,<object_name>]).
        <atomic_name>
                <property_type>
        END.
END.
```

There is no requirement that the original aggregate object cannot be generalized or specialized as well as the objects to which the disjunctive association is made. In the event that the aggregate object has more than one disjunctive association, only one small circle is drawn with all disjunctive association lines emanating from it. The disjunctive association is used when one and only one object of a group of objects can be associated with an aggregate object at one time. In the next example we assume that an INSTRUCTOR can either be a CIVILIAN or MILITARY but not both an further that both CIVILIAN and MILITARY both carry information not common to the other.

```
object name: INSTRUCTOR.
        description?:.
        property: IS_A (EMPLOYEE).
        description?:.
        property: SSN.
        description?:.
        key?: YES.
                property type: NUMERIC.
                numeric length: 9.
                numeric constraints: MUST_FILL.
        property: EMPLOYER_IS_MILITARY.
        description?:.
        key?: NO.
                property type: BOOLEAN.
                boolean format: T/F.
                boolean constraints: MUST_EXIST.
        property: DEGREE.
        description?:.
        key?: NO.
                property type: ENUMERATION.
                enumeration list: BS, MS, PHD, AA, NONE.
                enumeration constraints:.
        property: EMPLOYER EITHER (CIVILIAN,MILITARY).
        description?:.
        property: END.
```

From this example it can be deduced that both CIVILIAN and MILITARY instructors share a common property "DEGREE", but that their respective employer information is sufficiently different to warrant different aggregate objects.

Within CIVILIAN or EMPLOYEE there need not be any relations given since an inverse relation is implied by the disjunctive association. Each of these two objects must. however. share a common property definition. SSN in this case.

Since the disjunctive association implies a logical branch from the object to one of two or more other objects it is important that a discriminator property exist. This discriminator property (EMPLOYER_IS_MILITARY in this case) uniquely identifies which disjunctive association is applicable to any particular instance of the object's data members. Unfortunately. this discriminator property is not easily checked to ensure it adequately decides each alternative available. One possibility. not used in this proposed DDL, would be to prompt for the discriminator property immediately after the disjunctive association is defined. Even by knowing which of an aggregate object's properties is to act as a discriminator for a disjunctive association may not give good error checking. Obviously a boolean discriminator can only guide a branch to two objects but even this is misleading since the discriminator might be the combination of two or more atomic subobjects. For these reasons it is generally impossible to validate the selection properties of a discriminator.

It is also possible for one of the disjunctive objects to specify its owner as a generalized object by the IS_A operator. If this were the case then a dotted line would be drawn from the small circle instead of a solid line. Also, the generalized object would become a double rectangle as well.

5. The Large Circle

Quite arbitrarily. the large circle has been chosen to represent the overview object. Emanating from the overview object may be either dotted or solid lines. however. the disjunctive association of any member aggregate objects of the overview object cannot be discerned without expansion. The syntax is:

```
<object_name>.
        OVERVIEW_OF (<object_name>,<object_name>[.<object_name>]).
```

An overview object cannot contain any other constructs or properties. This means that the above syntax definition is not only minimal, it is the *only* way an overview object may be defined. If the overview construct is contained in an aggregate object a logical, rather than syntax, error will result. Once the overview statement is terminated (with the period), the next prompt will be for another object name as shown in the next example.

```
object name: ARMED_SERVICES.
     description?:.
     property: OVERVIEW_OF (USN. USAF, USCG, USMC. SHIPS,
                    PLANES, HELOS, TANKS, PERSONNEL).

object name:
```

Here we see that by logically grouping the nine aggregate objects under the artificial heading of ARMED_SERVICES, we both streamline and clarify the database schema assuming other overview objects exist.

B. DDL TO INGRES

Earlier it was mentioned that a "partial" mapping would be demonstrated from the DDL to INGRES' QUEL commands without covering every INGRES construct. Actually, GLAD assumes the functionality of relation control since INGRES supports the relational model and therefore considers relations to be contained within the data tables themselves without being explicitly defined. INGRES also gives no presentation of the database schema.

INGRES allows three important database administration functions not yet covered by the proposed GLAD DDL. These functions are:

* Define Subschema View (DEFINE VIEW)
* Attach Permissions to Attributes (DEFINE PERMIT)
* Define Storage Structure of Physical Data (MODIFY TO)

From the data definition standpoint, the only INGRES commands that are of immediate concern are the CREATE and DEFINE INTEGRITY ON commands. The CREATE command creates a table with a table name, the table's attributes (columns) and the attribute's formats. In effect, the table, in GLAD terminology, is an aggregate object without descriptions or relations. Furthermore, the aggregate object is actually a combination of both the CREATE and DEFINE INTEGRITY ON commands with additional features. First, the CREATE command is examined.

The syntax of the CREATE command is:

> ***create*** tablename(columnname=format{.columnname=format})
> [***with journaling***]

"Journaling" is an INGRES feature that, when specified, logs APPEND, REPLACE and DELETE commands made against the table. This feature is presently not supported under GLAD. "Tablename" is the same as GLAD's object name. The "columnname" is the same as GLAD's atomic object name. The "format" is an abbreviated form of GLAD's atomic property characteristics less any constraints. Soon we will see how a GLAD aggregate object is translated into corresponding INGRES constructs.

The DEFINE INTEGRITY ON syntax is:

define integrity on range_var ***is*** qual

For all intensive purposes, "range_var" is simply an abbreviation of a tablename specified by a RANGE OF QUEL command. "Qual" is a single variable qualification. This qualification is a constraint on one of the table's column names that allows greater than, less than, and the equals symbol to define a restricting value. For example:

```
/*  Allow no age in employee's table (e) to be less than 17 */

        define integrity on e is e.age >= 17
```

This constraint mechanism does not allow comparison with other attributes within the same row of the table but will allow range constraints by creating two such DEFINE INTEGRITY commands, each specifying one of the bounds. Obviously, the GLAD constraint mechanisms provide a much richer variety of constraints such that those constraints not supported by INGRES would have to be administered and monitored by GLAD. INGRES does not support any relations (regular, disjunctive association or specialized to generalized), overview objects, enumeration, boolean, numeric (however numeric can be usually be treated as an integer with few problems), or any specialized constraints not concerned with bounds or ranges (MUST_EXIST, JUSTIFY_L, CASE_INDEP, etc.).

Taking these disparities into consideration moves the GLAD system designer to tend to handle *all* integrity constraints and leave it to the underlying DBMS to store the data only. Assuming this to be the case, then even the DEFINE INTEGRITY ON command can be ignored leaving GLAD to rely only on the CREATE command to interface with INGRES. In other words, the only command that GLAD will issue to INGRES when creating the database schema is the QUEL CREATE command.

INGRES requires that the format of each column name be specified as one of the storage formats shown in figure 4.1.

GLAD can easily map each property constraint into one of INGRES' storage formats given the constraints. For example, if an integer is bounded by 10 and 72 then it would be assigned an INGRES format of "i1". If the precision of a floating point number is less than 17 it would be assigned a format of "f4". Three

Notation	Type	Range
c1-c255	Character	A string of 1 to 255 characters
text(1)-text(2000)	Text	A string of 1 to 2000 characters
i1	1-byte integer	-128 to +127
i2	2-byte integer	-32.768 to +32,767
i4	4-byte integer	-2.147,483.647 to + 2.147.483.647
f4	4-byte floating	-10**38 to +10**38 (7 decimal precision)
f8	8-byte floating	-10**38 to +10**38 (17 decimal precision)
date	date(12 bytes)	1-jan-1582 to 31-dec-2382 (for absolute dates) and -800 years to 800 years (for time intervals
money	money(8 bytes)	$-99999999999999.99 to $99999999999999.99

Figure 4.1 - INGRES Formats

GLAD property types do not fit nicely into the INGRES formats. The first is enumeration. Enumeration can be stored as an integer with the number of enumerated objects determining the integer byte size. For instance, if ten enumerated items were listed. then the storage format would be "i1". The reasoning behind this is that an enumerated object can assume only one value within a list ad by internally assigning an integer to each value, the value can be efficiently stored and converted by GLAD when needed. The second nonconforming type is the date. The date can be mapped into everyone of INGRES' date formats but an algorithm would need to be developed to match each of the possible GLAD date formats with each of INGRES' date formats. The last construct is numeric. This construct will pose no problems if it is stored as an integer.

With these conversion rules in mind, figure 4.2 shows the INGRES conversion statements for the GLAD schema given in figure 4.3.

Appendix D contains further examples of the necessary translation from GLAD to INGRES. Having created the basic data table skeleton, it will be up to the GLAD data manipulation language (DML) to interface with INGRES. The DML will utilize the QUEL APPEND, DELETE, REPLACE and RETRIEVE commands in much the same way the DDL has exploited the CREATE command.

By minimizing the interaction and responsibilities of the underlying DBMS it is envisioned that GLAD, once implemented, will be able to be quickly adapted to almost any underlying data model/DBMS.

```
/* Create the employee aggregate object */
create EMPLOYEE (SSN=i4, EMP.NAME=c15)

/* Create the faculty aggregate object */
create FACULTY (SSN=i4, TENURE=c4)
```

Figure 4.2 - INGRES Conversion

```
object name: EMPLOYEE.
description?: .
property: EMP.TYPE EITHER (FACULTY.SECRETARY,TECHNICIAN).
          description?: EMPLOYEE CAN BE ONE AND ONLY ONE
          property: BELONGS_TO LINE_TO (CAMPUS).
          description?: .
          property: SSN.
          key?: Y.
          description?: SOCIAL SECURITY NUMBER.
                    property type: NUMERIC.
                    numeric length: 9.
                    numeric constraints: MUST FILL.
          property: EMP.NAME.
          key?: N.
          description?: .
                    property type: TEXT.
                    text length: 15.
                    text constraints: MUST EXIST, JUSTIFY(R).
          property: END.

object name: FACULTY.
description?: .
          property: SSN.
          same as in object "employee"?: Y.
          key?: Y.
          property: IS_A (EMPLOYEE).
          description?: .
          property: TENURE.
          key?: N.
          description?: .
                    property type: ENUMERATION.
                    enumeration list: NONE, SOME, FULL.
                    enumeration constraints: CASE INDEPENDENT.
          property: DEPT_BELONGS_TO LINE_TO (DEPT).
          description?: .
          property: END.
```

Figure 4.3 - GLAD Sample Object Description

V. CONCLUSIONS AND RECOMMENDATIONS

The GLAD concept has been presented as the essential ingredient towards expanding the user population base of DBMS'. The proposed DDL has been shown to support GLAD's data model representation in all aspects except recursion. It is important to point out that the creation of the schema is but a small part of the database administrator's duties and a smaller part in the overall GLAD system. However, this thesis has shown that a small language can not only define complex interrelationships but it can also describe the integrity constraints to keep the data accurate.

A great number of problems still exist with the data definition subsystem. Some of the more prominent tasks include:

* Internal data structures of the metadata
* Graphics algorithms to implement schema representations
* Logic error checking
* Subschema definition and management

A great number of enhancements are also in order to GLAD's basic specifications. The enhancements might include such things as:

* An online Data Dictionary

* Multiple file manipulation

* Implementation of shorthand notation for the experienced user

* Adaptation packages to other DBMS'

* Interactive Help facility

The proposed DDL can provide the basic stepping stone in forwarding the progress in these areas. The most important conclusion from this thesis is that GLAD is not just desirable, it is possible. Much of the detailed analysis for GLAD has yet to be performed but the concepts are clear and the final product sorely needed in today's complex world.

APPENDIX A: FORMAL SPECIFICATION OF A DDL FOR GLAD

The following is BNF for the GLAD Data Definition Language. Square brackets [] are used to indicate optional constructs. The angled brackets <> enclose named constructs. The bar | can be read as "or" between two or more choices of constructions that will satisfy the composition of the construct.

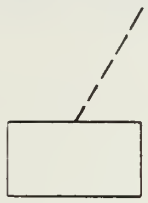
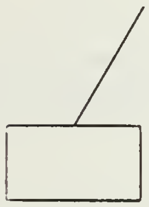
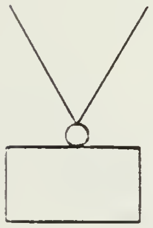
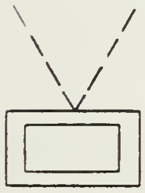
```
<database_schema>:=
            <object><property>[<property>]<end>[database_schema]<end>

<object>:=
            <object_name>[<desc>]

<object_name>:=
            <string>. | <end>

<string>:=<character> [<string>]

<character>:=
            <uc_letter> | <lc_letter> | <digit> | <special_char>

<lc_letter>:=
            a | b | c | ... | z

<uc_letter>:=
            A | B | C | ... | Z

<digit>:= 0 | 1 | 2 | ... | 9

<special_char>:=
            ! | @ | # | $ | % | ^ | & | * | ( | ) | _ | - | + | = | ? | : | ; | " | '

<property>:=
            <reg_rel_name> LINE_TO (<object_name>) [<desc>].
            |<disjunct_name>EITHER(<object_name>,<object_name>[,<object_name>])
            [<desc>].
            |IS_A (<object_name>)[<desc>].
            | OVERVIEW_OF (<object_name> [,<object_name>]) [<desc>].
            | <atomic_object> [<desc>].
            | <end>

<reg_rel_name>:=
            <string>

<disjunct_name>:=
            <string>

<atomic_object>:=
            <atomic_name>.<is_it_key>.<property_type>.

<atomic_name>:=
            <string>

<is_it_key>:=
            Y | N

<property_type>:=
            TEXT.<text_length>.[<text_const>].
            |INTEGER.[<int_range>].[<int_const>].
            |FLOAT.<float_prec>.[<float_range>].[<float_const>].
            |DATE.<date_format>.[<date_range>].[<date_const>].
            |MONEY.[<mony_range>].[<mony_const>].
            |ENUMERATION.<enum_list>.[<enum_const>].
            |BOOLEAN.<boolean_format>.[<boolean_const>].
            |NUMERIC.<num_length>.[<num_const>].

<text_length>:=
            <integer>.

<integer>:=
            <digit>[<digit>]

<text_const>:=
            <t_const>[.<t_const>]

<t_const>:=
            MUST_EXIST | MUST_FILL | NO_BLANKS | CASE_INDEP |
            ALL_LC | ALL_UC | NO_NUMERIC | PAD_BLANKS
```

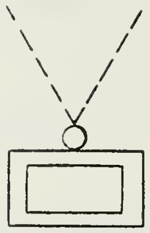
```
<int_range>:=
            [<sign>]<integer> TO [<sign>]<integer>
            | <relation_sign><integer>

<sign>:=  + | -

<int_const>:=
            <i_const>[,<i_const>]

<i_const>:=
            MUST_BE<relation_sign><atomic_name>[<connector><i_const>]

<connector>:=
            AND | OR

<relation_sign>:=
            < | <= | = | >= | > | <>

<float_prec>:=
            <integer>

<float_range>:=
            <float_num> TO <float_num>
            | <relation_sign> <float_num>

<float_num>:=
            [<sign>] <integer>.<integer>
            | <integer>
            | <float_num> E [<sign>] <integer>

<float_const>:=
            <f_const> [,<f_const>]

<f_const>:=
            MUST_BE<relation_sign><atomic_name>[<connector><f_const>]

<date_format>:=
            <date_comp> [<sep> <date_comp>]
```

```
<date_comp>:=
            DD | MM | MMM | MONTH | YY | YYYY

<sep>:=  - | / | . | blank

<date_range>:=
            <date> TO <date>
            | <relation_sign> <date>

<date>:=  <day> <month> <year>

<day>:=  1 | 2 | 3 | ... | 31

<month>:=
            1 | 2 | 3 | ... | 12

<year>:=  <digit> <digit> <digit> <digit>

<date_const>:=
            MUST_BE<relation_sign><atomic_name>[<connector><date_const>]

<mony_range>:=
            <money> TO <money>
            | <relation_sign> <money>

<money>:=
            [$] <integer> [.<digit><digit>]

<mony_const>:=
            MUST_BE<relation_sign><atomic_name>[<connector><mony_const>]

<enum_list>:=
            <string> [, <string>]

<enum_const>:=
            MUST_EXIST | ALLOW_ABBREV

<boolean_format>:=
            T/F | ON/OFF | Y/N | 0/1
```

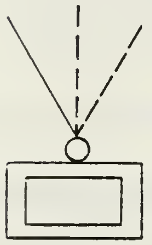
```
<boolean_const>:=
               MUST_EXIST

<num_length>:=
               [<integer>]

<num_constraints>:=
               <n_const> [,<n_const>]

<n_const>:=
               MUST_EXIST | MUST_FILL | PAD_BLANKS | PAD_ZEROES

<desc>:=       <character>[<string>] | blank [<string>]

<end>:=        END. | . | return key
```

APPENDIX B: GLAD DIAGRAM INTERPRETATION

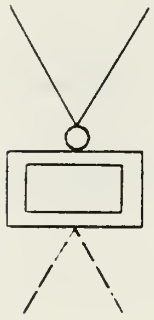
DIAGRAM	*INTERPRETATION*
	This is a specialized object being joined to its generalized object
	Normal relation from an object
	This object has a disjunctive association with other objects
	This is a generalized object where the dotted lines join it with the specialized components



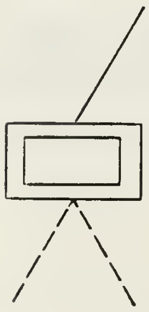
This is a generalized object that has specified a disjunctive association with its specialized components



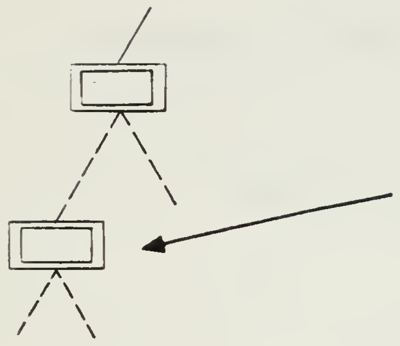
Two of the objects of the disjunctive association are specialized objects while the other is not



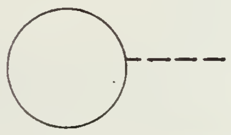
A generalized object that has a disjunctive association with objects that are not its specialized components



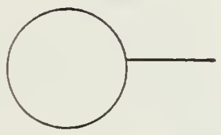
The solid line represents a normal relation from a generalized object



This generalized object is a specialized object as well

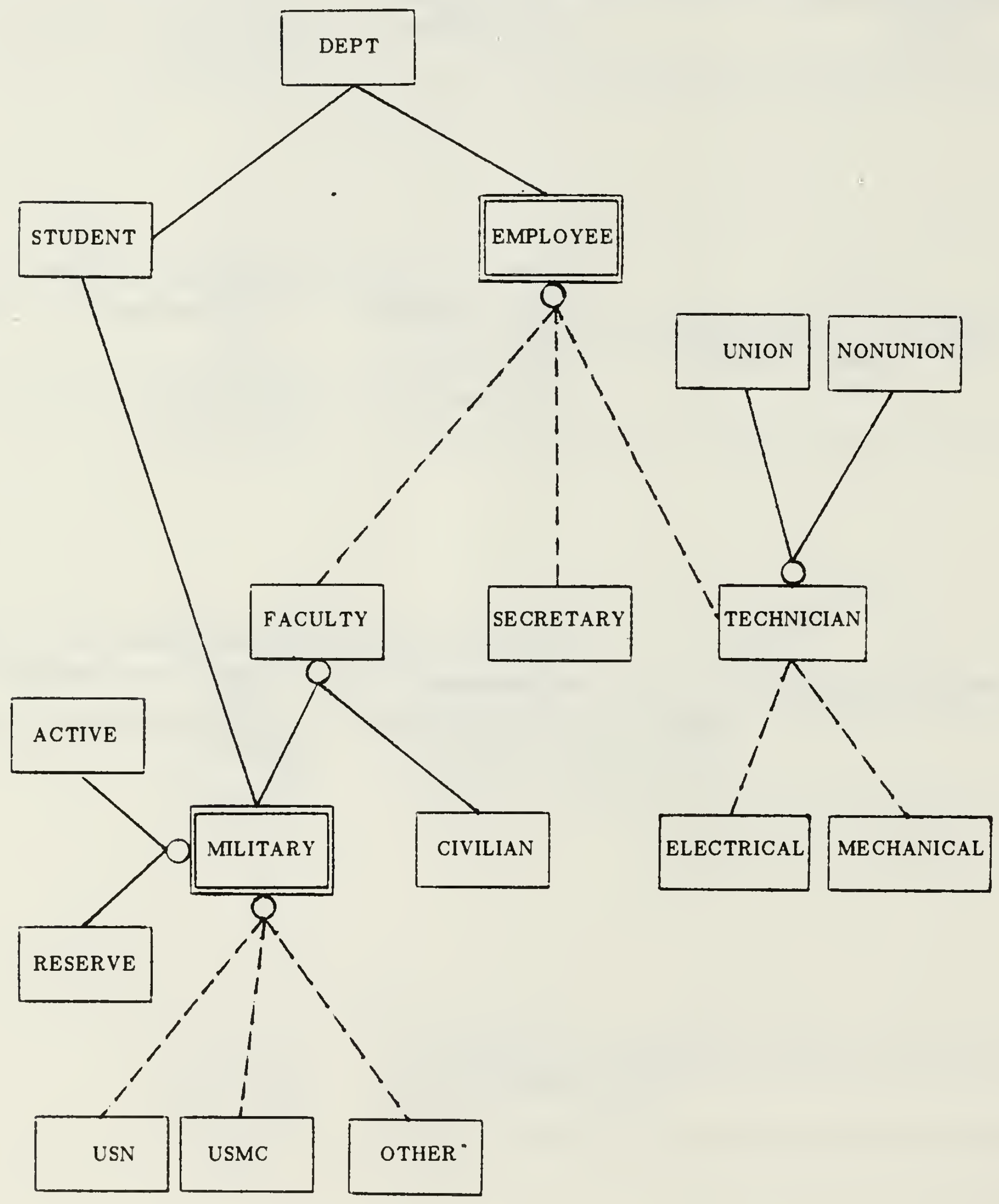


Overview object that contains an object that is either specialized or generalized to another object



Overview object to another object that it shares neither specialized or generalized relations

APPENDIX C : EXAMPLE CONCEPTUAL SCHEMA



APPENDIX D : SAMPLE SESSION

PART I: Using the DDL to Create Schema

```
object name: DEPT.
desc?: SCHOOL DEPARTMENT.
        property: DEPT_NAME.
        desc?: .
        key?: YES.
                property type: TEXT.
                text length : 5.
                text constraints: MUST_EXIST.
        property: CHAIRMAN.
        desc?: .
        key?: NO.
                property type: TEXT.
                text length : 15.
                text constraints: NO_NUMERIC.
        property: DEPT_LOC.
        desc?: .
        key?: NO.
                property type: TEXT.
                text length : 25.
                text constraints: .
        property: END.
```

```
object name: STUDENT.
desc?: STUDENT ATTENDING XYZ MILITARY SCHOOL.
          property: SSN.
          desc?: .
          key?: YES.
                    property type: NUMERIC.
                    numeric length: 9.
                    numeric constraints: MUST_FILL.
          property: FIRST_NAME.
          desc?: .
          key?: NO.
                    property type: TEXT.
                    text length : 15.
                    text constraints: CASE_INDEP.
          property: LAST_NAME.
          desc?: .
          key?: N.
                    property type: TEXT.
                    text length : 25.
                    text constraints: MUST_EXIST,NO_NUMERIC.
          property: DEPT_MAJOR.
          desc?: .
          key?: NO.
                    property type: TEXT.
                    text length : 5.
                    text constraints: MUST_EXIST.
          property: BOS.
          desc?: BRANCH OF SERVICE.
          key?: NO.
                    property type: ENUMERATION.
                    enumerated list: USN, USMC, OTHER.
                    enumeration constraints: MUST_EXIST.
          property: MILITARY_DATA LINE_TO (MILITARY).
          desc?: .
          property: END.
```

```
object name: EMPLOYEE.
desc?: EMPLOYEE OF XYZ MILITARY SCHOOL.
          property: SSN.
          same as in "student"?: YES.
          key?: YES.
          property: DEPT.
          desc?: .
          key?: NO.
                    property type: TEXT.
                    text length : 5.
                    text constraints: .
          property: FIRST_NAME.
          same as in "student"?: YES.
          key?: NO.
          property: LAST_NAME.
          same as in "student"?: YES.
          key?: NO.
          property: JOB_TITLE.
          desc?: .
          key?: NO.
                    property type: ENUMERATION.
                    enumerated list: FACULTY, SECRETARY, TECHNICIAN.
                    enumeration constraints: MUST_EXIST.
          property: BELONGS_TO LINE_TO (DEPT).
          desc?: .
          property: TYPE_EMPLOYEE EITHER (FACULTY,
                              SECRETARY, TECHNICIAN).
          desc?: .
          property: END.
```

```
object name: TECHNICIAN.
desc?: LAB TECHNICIAN.
          property: SSN.
          same as in "student"?: YES.
          key?: YES.
          property: TECH_RATING.
          desc?: .
          key?: NO.
                    property type: TEXT.
                    text length : 4.
                    text constraints: PAD_BLANKS.
          property: SAFETY_QUAL?.
          desc?: .
          key?: NO.
                    property type: BOOLEAN.
                    boolean format: Y/N.
                    boolean constraints: MUST_EXIST.
          property: UNION?.
          desc?: .
          key?: NO.
                    property type: BOOLEAN.
                    boolean format: Y/N.
                    boolean constraints: MUST_EXIST.
          property: IS_A (EMPLOYEE).
          desc?: .
          property: AFFILIATION EITHER (UNION, NONUNION).
          desc?: .
          property: END.
```

```
object name: UNION.
desc?: LABOR UNION.
          property: SSN.
          same as in "student"?: YES.
          key?: YES.
          property: UNION_NAME.
          desc?: .
          key?: NO.
                    property type: TEXT.
                    text length : 25.
                    text constraints: MUST_EXIST.
          property: CHAPTER.
          desc?: .
          key?: NO.
                    property type: TEXT.
                    text length : 10.
                    text constraints: .
          property: DATE_JOINED_UNION.
          desc?: .
          key?: NO.
                    property type: DATE.
                    date format: YY-MMM-DD.
                    date range: .
                    date constraints: .
          property: END.
```

```
object name: NON_UNION.
desc?: EMPLOYEE'S NON UNION INFO.
          property: SSN.
          same as in "student"?: YES.
          key?: YES.
          property: DATE_LAST_EMPLOYMENT.
          desc?: .
          key?: NO.
                    property type: DATE.
                    date format: YY/MMM/DD.
                    date range: >= 16 JAN 1957.
                    date constraints: MUST_BE >= DATE_JOINED_SCHOOL.
          property: DATE_JOINED_SCHOOL.
          desc?: .
          key?: NO.
                    property type: DATE.
                    date format: YY/MMM/DD.
                    date range: >= 16 JAN 1957.
                    date constraints: MUST_ENTER.
          property: END.
```

```
object name: ELECTRICAL.
desc?: WORKERS ELECTRICAL QUALIFICATIONS.
        property: SSN.
        same as in "student"?: YES.
        key?: YES.
        property: SCHOOL_ATTENDED.
        desc?: .
        key?: NO.
                property type: TEXT.
                text length : 25.
                text constraints: JUSTIFY_R, CASE_INDEP.
        property: INSPECTOR_QUAL?.
        desc?: .
        key?: NO.
                property type: BOOLEAN.
                boolean format: Y/N.
                boolean constraints: .
        property: IS_A (TECHNICIAN).
        desc?: .
        property: END.
```

```
object name: MECHANICAL.
desc?: WORKERS NON ELECTRICAL QUALIFICATIONS.
        property: SSN.
        same as in "student"?: YES.
        key?: YES.
        property: SKILL_LEVEL.
        desc?: .
        key?: NO.
                property type : INTEGER.
                integer range: 1 TO 15.
                integer constraints: .
        property: EXPERTISE_AREA.
        desc?: .
        key?: NO.
                property type: ENUMERATION.
                enumerated list: PLUMBING, WELDING,
                                PAINTING, GENERAL.
                enumeration constraints: .
        property: IS_A (TECHNICIAN).
        desc?: .
        property: END.
```

```
object name: SECRETARY.
desc?: EMPLOYEE'S SECRETARIAL SKILLS.
          property: SSN.
          same as in "student"?: YES.
          key?: YES.
          property: WPM.
          desc?: WORDS PER MINUTE.
          key?: NO.
                    property type : INTEGER.
                    integer range: 1 TO 200.
                    integer constraints: .
          property: BONDED?.
          desc?: .
          key?: NO.
                    property type: BOOLEAN.
                    boolean format: T/F.
                    boolean constraints: .
          property: MACHINE_QUALIFIED.
          desc?: .
          key?: NO.
                    property type: ENUMERATION.
                    enumerated list: DS102. OJ90, HH90, 5656, DEC101,
                                        IBM740.
                    enumeration constraints: .
          property: IS_A (EMPLOYEE).
          desc?: .
          property: END.
```

```
object name: FACULTY.
desc?: EMPLOYEE'S FACULTY QUALIFICATIONS.
        property: SSN.
        same as in "student"?: YES.
        key?: YES.
        property: POSITION.
        desc?: .
        key?: NO.
                property type: TEXT.
                text length : 15.
                text constraints: .
        property: OFFICE_NO.
        desc?: .
        key?: NO.
                property type : INTEGER.
                integer range: 100 TO 1000.
                integer constraints: .
        property: TENURE.
        desc?: .
        key?: NO.
                property type: ENUMERATION.
                enumerated list: NONE. ASSOCIATE. ASSISTANT,
                                FULL.
                enumeration constraints: .
        property: IS_A (EMPLOYEE).
        desc?: .
        property: OCCUPATION EITHER (MILITARY. CIVILIAN).
        desc?: .
        property: END.
```

```
object name: MILITARY.
desc?: MEMBER'S MILITARY DATA.
          property: SSN.
          same as in "student"?: YES.
          key?: YES.
          property: ENTRY_DATE.
          desc?: .
          key?: NO.
                    property type: DATE.
                    date format: YY-MMM-DD.
                    date range: 10 JULY 1958 TO 11 JULY 2002.
                    date constraints: .
          property: BOS.
          same as in "student"?: YES.
          key?: NO.
          property: RANK.
          desc?: .
          key?: NO.
                    property type: ENUMERATION.
                    enumerated list: 01. 02. 03. 04. 05. 06. NA.
                    enumeration constraints: .
          property: STATUS.
          desc?: .
          key?: NO.
                    property type: ENUMERATION.
                    enumerated list: ACTIVE. RESERVE.
                    enumeration constraints: .
          property: ATTENDING_SCHOOL LINE_TO (STUDENT).
          desc?: .
          property: DUTY_STATUS EITHER (ACTIVE_DUTY,
                     RESERVE).
          desc?: .
          property: SERVICE_BRANCH EITHER (USN, USMC,
                     OTHER).
          desc?: .
          property: END.
```

```
object name: ACTIVE_DUTY.
desc?: .
          property: SSN.
          same as in "student"?: YES.
          key?: YES.
          property: DAYS_LEAVE.
          desc?: .
          key?: NO.
                    property type : INTEGER.
                    integer range: -99 TO 150.
                    integer constraints: .
          property: BASE_PAY.
          desc?: .
          key?: NO.
                    property type: MONEY.
                    money range: 0 TO 5090.78
                    money constraints: .
          property: TIME_LEFT_ON_CONTRACT:
          desc?: .
          key?: NO.
                    property type : INTEGER.
                    integer range: 0 TO 5000.
                    integer constraints: .
          property: END.
```

```
object name: RESERVE.
desc?: .
          property: SSN.
          same as in "student"?: YES.
          key?: YES.
          property: LAST_DRILL_DATE.
          desc?: .
          key?: NO.
                    property type: DATE.
                    date format: YY-MMM-DD.
                    date range: 10 JULY 1958 TO PRESENT.
                    date constraints: .
          property: RETIREMENT_POINTS.
          desc?: .
          key?: NO.
                    property type: FLOAT.
                    number of characters to the right of decimal point: 2.
                    floating point range: 0 TO 99.9.
                    floating point constraints: .
          property: END.
```

```
object name: USN.
desc?: .
        property: SSN.
        same as in "student"?: YES.
        key?: YES.
        property: RATING.
        desc?: .
        key?: NO.
                property type: TEXT.
                text length : 7.
                text constraints: .
        property: LAST_TOUR_TYPE.
        desc?: .
        key?: NO.
                property type: ENUMERATION.
                enumerated list: UNACCOMPANIED,
                                 ACCOMPANIED, HARDSHIP.
                enumeration constraints: MUST_EXIST.
        property: IS_A (MILITARY).
        desc?: .
        property: END.
```

```
object name: USMC.
desc?: .
        property: SSN.
        same as in "student"?: YES.
        key?: YES.
        property: MOS.
        desc?: .
        key?: NO.
                property type : INTEGER.
                integer range: 0 TO 9999.
                integer constraints: .
        property: LAST_RUC.
        desc?: LAST REPORTING UNIT CODE.
        key?: NO.
                property type : INTEGER.
                integer range: 0 TO 9999.
                integer constraints: .
        property: IS_A (MILITARY).
        desc?: .
        property: END.
```

```
object name: OTHER.
desc?: MILITARY MEMBER IS NEITHER USN OR USMC.
        property: SSN.
        same as in "student"?: YES.
        key?: YES.
        property: BRANCH.
        desc?: .
        key?: NO.
                property type: TEXT.
                text length : 10.
                text constraints: .
        property: ARMED?.
        desc?: .
        key?: NO.
                property type: BOOLEAN.
                boolean format: Y/N.
                boolean constraints: MUST_EXIST.
        property: IS_A (MILITARY).
        desc?: .
        property: END.


object name: CIVILIAN.
desc?: .
        property: SSN.
        same as in "student"?: YES.
        key?: YES.
        property: ADDRESS.
        desc?: .
        key?: NO.
                property type: TEXT.
                text length : 25.
                text constraints: PAD_BLANKS.
        property: ID_NO.
        desc?: .
        key?: NO.
                property type : NUMERIC.
                numeric length: 6.
                numeric constraints: .
        property: END.

object name: END.
```

PART II: INGRES Statements to Store Data

/*Create **STUDENT** aggregate object*/

```
create STUDENT (SSN=i4.FIRST_NAME=c15,LAST_NAME=c25,
DEPT_MAJOR=c5,BOS=i1)
```

/*Create **DEPT** aggregate object*/

```
create DEPT (DEPT_NAME=c5,CHAIRMAN=c15,DEPT_LOC=c25)
```

/*Create **EMPLOYEE** aggregate object*/

```
create EMPLOYEE (SSN=i4,DEPT=c5,FIRST_NAME=c15,
LAST_NAME=c25,JOB_TITLE=i1)
```

/*Create **TECHNICIAN** aggregate object*/

```
create TECHNICIAN (SSN=i4,TECH_RATING=c4,SAFETY_QUAL?=i1,
UNION?=i1)
```

/*Create **UNION** aggregate object*/

```
create UNION (SSN=i4,UNION_NAME=c25,CHAPTER=c10,
DATE_JOINED_UNION=date)
```

/*Create **NON_UNION** aggregate object*/

```
create NON_UNION (SSN=i4,DATE_LAST_EMPLOYED=date,
DATE_JOINED_SCHOOL=date)
```

/*Create **ELECTRICAL** aggregate object*/

```
create ELECTRICAL (SSN=i4,SCHOOL_ATTENDED=c25,
INSPECTOR_QUAL?=i1)


/*Create MECHANICAL aggregate object*/

create MECHANICAL (SSN=i4.SKILL_LEVEL=i1.EXPERTISE_AREA=i1)


/*Create SECRETARY aggregate object*/

create SECRETARY (SSN=i4,WPM=i2.BONDED?=i1.
MACHINE_QUALIFIED=i1)


/*Create FACULTY aggregate object*/

create FACULTY (SSN=i4.POSITION=c15.OFFICE_NO=i2.TENURE=i1)


/*Create MILITARY aggregate object*/

create MILITARY (SSN=i4,ENTRY_DATE=date,BOS=i1.RANK=i1,
STATUS=i1)


/*Create ACTIVE_DUTY aggregate object*/

create ACTIVE_DUTY (SSN=i4,DAYS_LEAVE=i2,BASE_PAY=money,
TIME_LEFT_ON_CONTRACT=i2


/*Create RESERVE aggregate object*/

create RESERVE (SSN=i4.LAST_DRILL_DATE=date.
RETIREMENT_POINTS=f4)


/*Create USN aggregate object*/

create USN (SSN=i4.RATING=c7.LAST_TOUR_TYPE=i1)


/*Create USMC aggregate object*/
```

```
create USMC (SSN=i4,MOS=i2,LAST_RUC=i2)

/*Create OTHER aggregate object*/

create OTHER (SSN=i4,BRANCH=c10,ARMED?=i1)

/*Create CIVILIAN aggregate object*/

create CIVILIAN (SSN=i4,ADDRESS=c25,ID_NO=i4)
```

APPENDIX E: PROMPT FLOW AND SUGGESTED HELP DATA WITH PROMPTS

PROMPT[1]	DISPLAYED CHOICES[2]	NEXT PROMPT
object name:	[object]	property:
	END	Return to calling environment
	(All previously defined object names)[3]	
property:	[name] LINE_TO [object][4]	property:
	[name] EITHER [objects]	.
	IS_A [object]	.
	OVERVIEW_OF[objects]	property:
	(All previously defined property names)	same as in "x"?:
property type:	TEXT	text length:
	INTEGER	integer range:
	FLOAT	floating point precision:
	DATE	date format:
	MONEY	money range:
	ENUMERATION	enumerated list:
	BOOLEAN	boolean format:
	NUMERIC	numeric length:
text length:	[integer]	text constraints:
integer range:	[integer] TO [integer]	integer constraints:
	[<,<=,=,<>,>,>=] [integer]	integer constraints:
floating point precision:	[float] TO [float]	floating point range
date format:	DD[/-.]YY[/-.]MM	date range:
	DD[/-.]MONTH[/-.]YYYY	
	MMM[/-.]DD[/-.]YYYY	
	etc	
money range:	[money] TO [money]	money constraints:
	[<,<=,=,<>,>,>=] [money]	money constraints:

[1]The desc prompt follows object and property but is ignored here.

[2]The first displayed choice(s) is normally the required syntax so it can not be selected.

[3]Cannot be selected since selection is confined to unique names.

PROMPT	DISPLAYED CHOICES	NEXT PROMPT
enumerated list:	[string],[strings]	enumerated constraints:
boolean format:	T/F ON/OFF Y/N 0/1	boolean constraints: . . boolean constraints:
numeric length:	[integer]	numeric constraints:
text constraints:	MUST_EXIST MUST_FILL NO_BLANKS CASE_INDEP ALL_LC	property . . . .
	ALL_UC	property:
integer constraints	MUST_BE [<,<=,=,<>,>,>=] [object] AND | OR	property: integer constraints:
floating point range:	[float] TO [float] [<,<=,=,<>,>,>=] [float]	float constraints: float constraints:
date range:	[date] TO [date] [<,<=,=,<>,>,>=] [date]	date constraints: date constraints:
money constraints	MUST_BE [<,<=,=,<>,>,>=] [object] AND | OR	property: money constraints:
enumerated constraints:	MUST_EXIST ALLOW_ABBREV	property: property:
boolean constraints:	MUST_EXIST	property:
numeric constraints:	MUST_EXIST MUST_FILL PAD_BLANKS PAD_ZEROES	property . . property:
float constraints	MUST_BE [<,<=,=,<>,>,>=] [object] AND | OR	property: float constraints:
date constraints	MUST_BE [<,<=,=,<>,>,>=] [object] AND | OR	property: date constraints:

[4]The following four syntax may be selected to display all previously defined object names.

LIST OF REFERENCES

[BRAG84] Bragger, R. P., A. Dudler, J. Rebsamen, and C. A. Zehnder, "Gambit: An Interactive Database Design Tool for Data Structures, Integrity Constraints and Transactions", *Proceedings of IEEE International Conference on Data Engineering,* (Los Angeles),pp 399-407, 1984.

[CHEN76] Chen, P. P., "The Entity-relationship Model: Towards a Unified View of Data", *ACM Transactions on Data Base Systems,* Vol. 1, No. 1, 1976.

[CODD77] Codd, E. F., "A Relational Model of Data for Large Shared DataBanks", *Communications of the ACM,* Vol. 13, No. 6, June 1970.

[KROE83] Kroenke, David, *Database Processing* p. 6, Science Research Associates, Inc., 1983.

[PETR76] Petrick, S. R., "On Natural Language Based Computer Systems", *IBM Journal on Research Developments,* Vol. 20, No. 4, pp 314-325, July 1976.

[TSIC78] Tsichritzis, D., and A. Klug, "*The ANSI/X3/SPARC DBMS Framework Report of the Study Group on Database Management Systems*", Pergamon Press, Oxford, 1978.

[WU85] Wu, C. T., "A Unified Interface Method for Interacting with a Database", Submitted for publication, 1985.

[ZLOO77] Zloof, M. M., "Query-by-Example: A Database Language", *IBM Systems Journal,* No. 4, pp 324-343, Dec 1977.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5002	2
2.	Chairman, Code 52 Department of Computer Science Naval Postgraduate School Monterey, California 93943	2
3.	Computer Technology Programs, Code 37 Naval Postgraduate School Monterey, California 93943	1
4.	C. T. Wu, Code 52Wq Department of Computer Science Naval Postgraduate School Monterey, California 93943	5
5.	Capt. K. L. Wartick, USMC 1541 Cedar Street Niagara, Wisconsin 54151	3
6.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2

18070 2